SCA4 – An Evolved Framework

Chalena M. Jimenez, Kevin W. Richardson, and Donald R. Stephens
Joint Tactical Radio System

***Abstract* — It has been over 10 years since the first iteration of Software Communication Architecture (SCA) and more than 5 years since the release of the previous version of the specification, SCA 2.2.2. While there have been numerous technological advancements and lessons learned in the field of software defined radio since the release of SCA 2.2.2, the framework remained relatively static and was extended to include a small subset of the new features. The latest specification release, commonly known as SCA4, incorporates a wider range of resolutions, significantly optimizes the framework and improves a programmer's ability to develop software defined radios efficiently. The new standard builds on a decade of expertise and makes the SCA even more relevant in today's market of resource constrained systems with the ever increasing need for secure mobile communications. With the breadth of potential SCA based target platforms and applications, SCA4 broadens its applicability beyond U.S. military software defined radios.**

***Index Terms*—[SCA, SDR, JTRS, Android]**

I. Introduction

THE SCA is an open-architecture specification that defines the interactions between software applications and hardware platforms. In Joint Tactical Radio System (JTRS), these software applications are waveforms and the hardware platforms are radios. The SCA framework has guided the development and evolution of the software defined radio domain and its concepts have been used within multiple industries, products and countries beyond the US DoD community.

A primary goal of military software defined radios is to minimize the amount of effort required in porting waveform applications to different radio platforms – the SCA establishes the infrastructure to achieve this objective. As illustrated in Figure 1, an SCA-based software defined radio provides a standardized infrastructure for software deployment and configuration; while ensuring interoperability between SCA-based products. SCA components may be extended by the JTRS Application Program Interfaces (APIs) to provide platform specific capabilities. The SCA and JTRS APIs promote waveform portability and reuse by isolating the waveform application from the radio set.

The modular nature of SCA4 builds in flexibility for the evolution of SCA-compliant products and the standard itself as technology and requirements change. This newest framework emphasizes flexibility & scalability throughout the specification. From a system developer perspective, the flexibility can be used to innovate and provide solutions which are appropriately tailored to a particular product. For the radio user, the flexibility permits customization and extension of the features and capabilities of the original product.

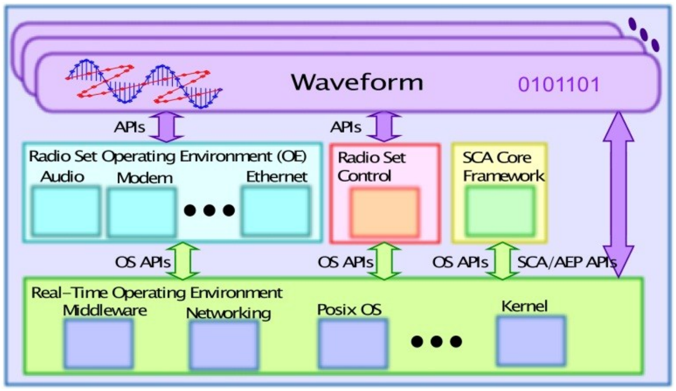


Figure 1 Architecture of SCA-Enabled Radio

II. Push Model

The SCA was originally developed with a client-side 'pull' design pattern which required a multi-step approach to deploying components in the domain. In SCA 2.2.2 application components register with the naming service upon entry to the domain and then the *ApplicationFactory* queries the naming service to discover when application components became available. Only after the components have registered could the *ApplicationFactory* continue the deployment process. Use of a vulnerable naming service is no longer supported in SCA4 and instead replaces it with a 'push' model approach, in which an application component is provided access to an instantiation of the standalone *ComponentRegistry* interface that is associated with an ApplicationFactoryComponent. The application component registers with the *ComponentRegistry* instantiation and provides all of its information upfront with a single call. This change in the interaction model can achieve real reductions on boot-up time, perhaps a 50% decrease. An additional 'push model' benefit is that, unlike the 'pull' model approach, it does not allow access to vulnerable system data and eliminates the possibility of clients requesting information they should not have.

III. SCA Port Connections

A component architecture identifies endpoints or connection point between the individual software components of a software infrastructure or application. The SCA defines these endpoints as ports which are similar to a socket connection in a hardware domain. Illustrated in Figure 2, an

SCA port is a software address that represents a connection point for a component.

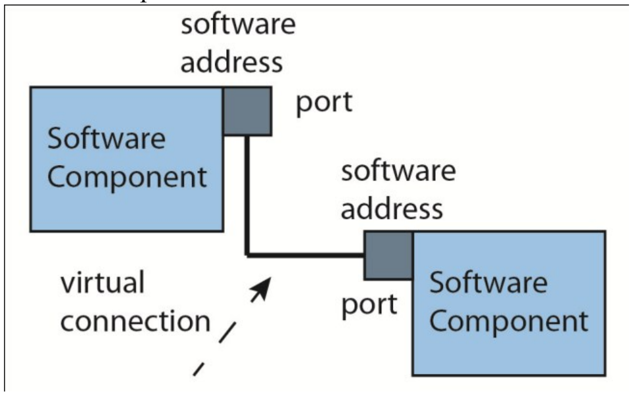


Figure 2 SCA Ports are addresses

Associated with a port is an interface (which can be an individual interface or an aggregation of interfaces) as illustrated in Figure 3. Knowledge of the address itself is insufficient – the communicating component must have knowledge of the interface. In SCA4, that information is provided in XML files for every component within the system. The specific system architecture determines whether the information is publicly accessible or whether communicating components must know the interface a priori.

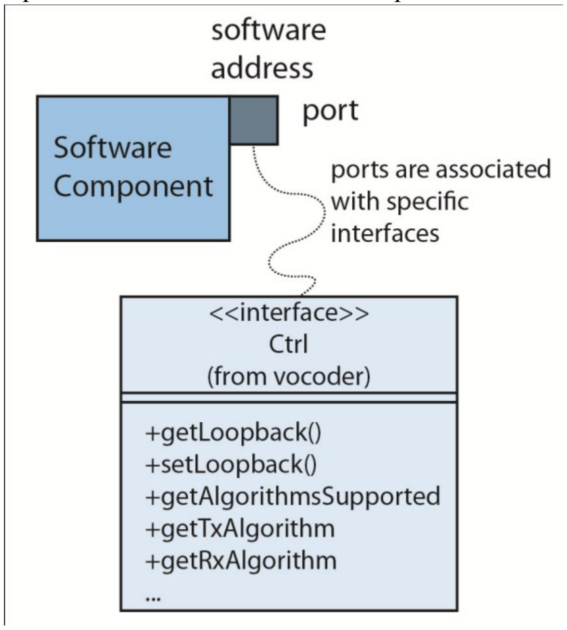


Figure 1 SCA Ports have an associated interface

SCA4 introduces a new static ports feature. Static ports allow for an implementation specific approach to connection establishment. Connections can be formed in an efficient manner at run time or at build time by providing a static predefined address for the connection. The impact of static ports is minimal for applications with a limited number of ports, but the capability will result in substantially more savings for systems with applications that require hundreds of port connections.

IV. Optional Inheritance

JTRS SCA-based products support a vast range of functionality and features. The original SCA defines a fairly rigid infrastructure which has been successfully implemented and deployed in software defined radios today. SCA4 augments the current capabilities by providing SCA infrastructures with the ability to be reconfigured such that they better align with the maintenance and development needs of existing and future software defined radios. The majority of the modifications made to the interfaces in SCA4 are not significant departures from their earlier counterparts yet they are essential elements of a more adaptive architecture. The potential impact of affecting the existing interfaces was considered during the development process, and a determination was made that the benefits outweighed the cost. Fairly simplistic tools could offset the costs associated with migrating to the new interfaces by providing an offline translation service to align existing SCA products with the new interfaces.

For example Figure 4 depicts how the *DomainManager* interface changed between SCA 2.2.2 and SCA4. The majority of the operations and attributes maintain a one-to-one mapping between the versions, specifically items 1-3 and 6-7 in the example shown in Figure 4. The primary underlying rationale for the interface change was to introduce a least privilege pattern at the framework level. Least privilege allows for a bevy of alternatives in terms of how one lays-out the components within their system and what information is readily available to external agents. For example, the registration methods were moved into new standalone interfaces, indicated by items 4 and 5 in Figure 4. The introduction of these new interfaces allows the product developer to determine which additional information is essential to provide to external clients.

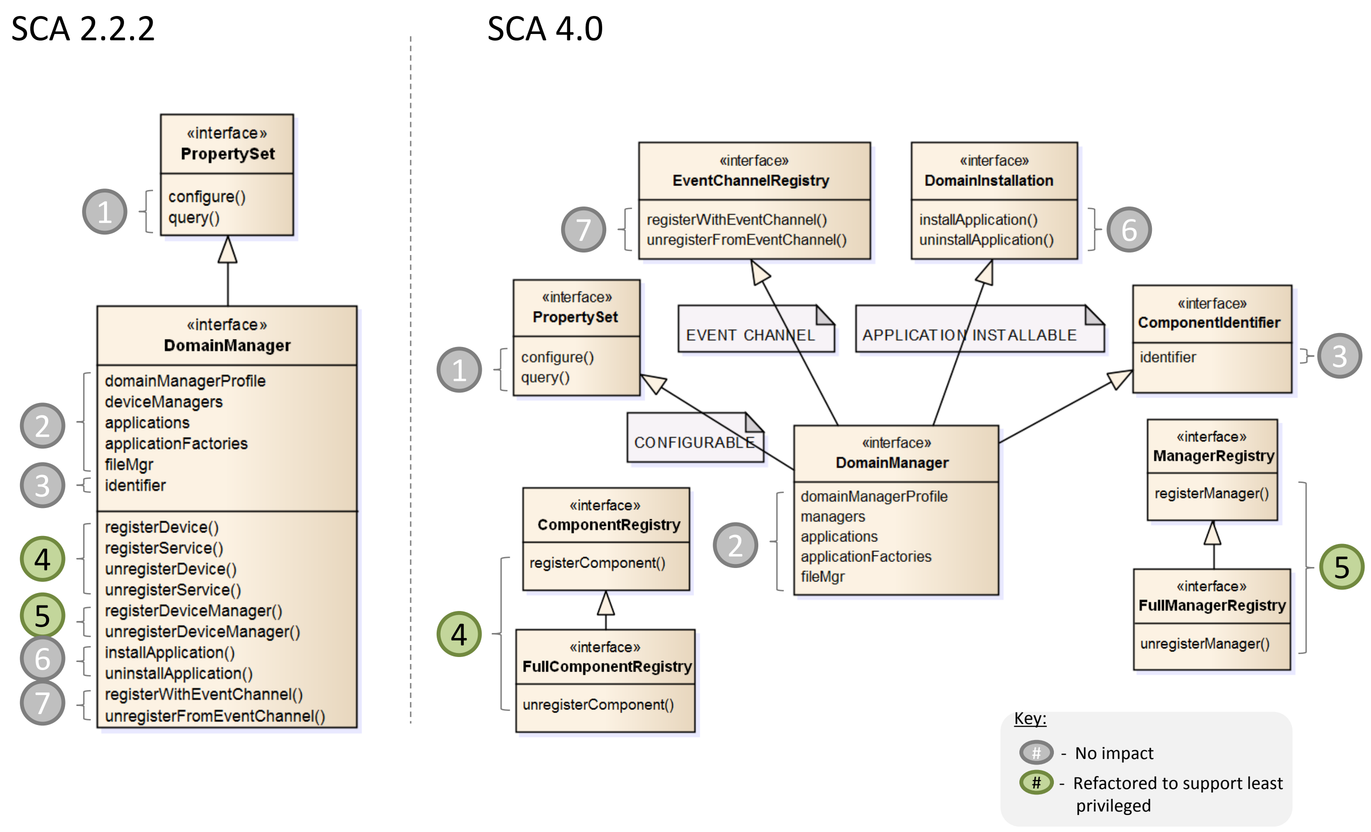
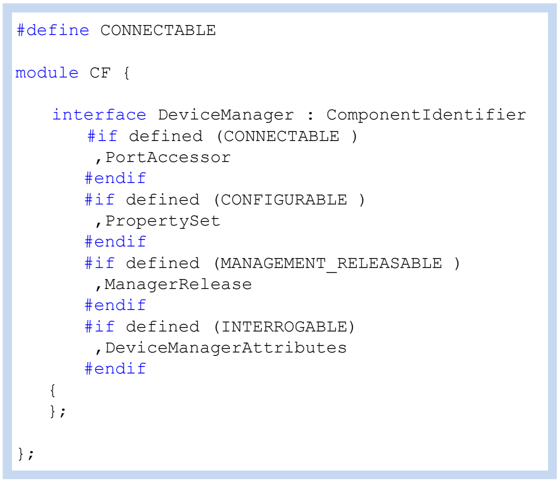


Figure 4 – *DomainManager* Interface Transformation

Previously, the developer was required to implement all of the inherited interfaces even though all of those interfaces may not have been necessary. For example, an SCA 2.2.2 *Resource* interface inherits the *TestableObject* interface and needs to implement the *runTest* operation regardless of whether or not the component provides a test capability. SCA4 allows the developer to only include the interfaces necessary for the implementation, eliminating unused or underutilized code. With SCA4, the developer would not include *TestableObject* within the interface inheritance hierarchy and not be required to implement a *runTest* operation. The SCA4 can lower the cost of software defined radios with the new optional inheritance technology, which reduces software development and maintenance.

Optional inheritance is carried out in SCA4 via directives in the IDL definitions. Each directive is associated with a Unit of Functionality (UOF) which is a grouping of requirements that provides a particular set of functionality. Figure 5 provides a sample excerpt from a generic DeviceManager.idl and shows how the directives can be utilized in an implementation. In this example, the Connectable UOF is enabled (i.e. via #define CONNECTABLE) which results in the *DeviceManager* interface extending the *PortAccessor* interface. The other UOFs are not enabled and therefore the *DeviceManager* interface would not extend any of the other optional interfaces (i.e. *PropertySet*, *ManagerRelease*, and *DeviceManagerAttributes*) indicated by grayed-out interfaces in Figure 5. Optional inheritance's benefit is the reduction in the number of applicable requirements; however one should not lose sight of the fact that the savings associated with this feature are distributed across the entire software development life cycle.



```
#define CONNECTABLE

module CF {

    interface DeviceManager : ComponentIdentifier
       #if defined (CONNECTABLE )
        ,PortAccessor
       #endif
       #if defined (CONFIGURABLE )
        ,PropertySet
       #endif
       #if defined (MANAGEMENT_RELEASABLE )
        ,ManagerRelease
       #endif
       #if defined (INTERROGABLE)
        ,DeviceManagerAttributes
       #endif
    {
    };

};
```

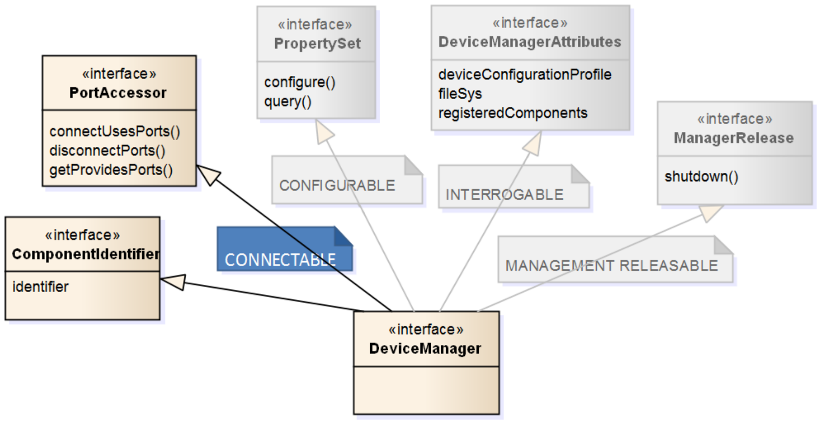


Figure 5 – Optional Inheritance Directives Example

V. Component Model

SCA4 introduces a component model that benefits specification users primarily from a system engineering perspective. One of the comments targeted to earlier SCA versions was that it was difficult to understand because they provided "interface centric" system views. The component model addresses this concern by providing a clear separation between interfaces (an element that defines "what" needs to be done or "why" something needs to be done) and components (a modular, replaceable part of a system within a defined environment that encompasses both static and dynamic behaviors or "how" something is done). A notional representation of the hierarchy of the significant SCA components is shown in Figure 6.

The introduction of the component model does not imply significant changes to the existing requirements set. Many of the requirements in previous versions of the SCA were behavior based requirements – i.e. requirements that went beyond the interface level. These behavioral requirements now appear in a corresponding component in SCA4. An example of this would be the state transition requirements previously found in the *Device* interface are now located within the DeviceComponent sections of SCA4. It is important that the distinction between interface and behavioral requirements be made more evident – this adds usability to the SCA without incurring great cost on the developer side.

The incorporation of components provides a concrete bridge from interface to implementation which will become more and more important as additional optional capabilities are introduced into the specification. In addition, properly-developed, component will improve the prospect of enhanced portability and reuse of detailed architectural artifacts.

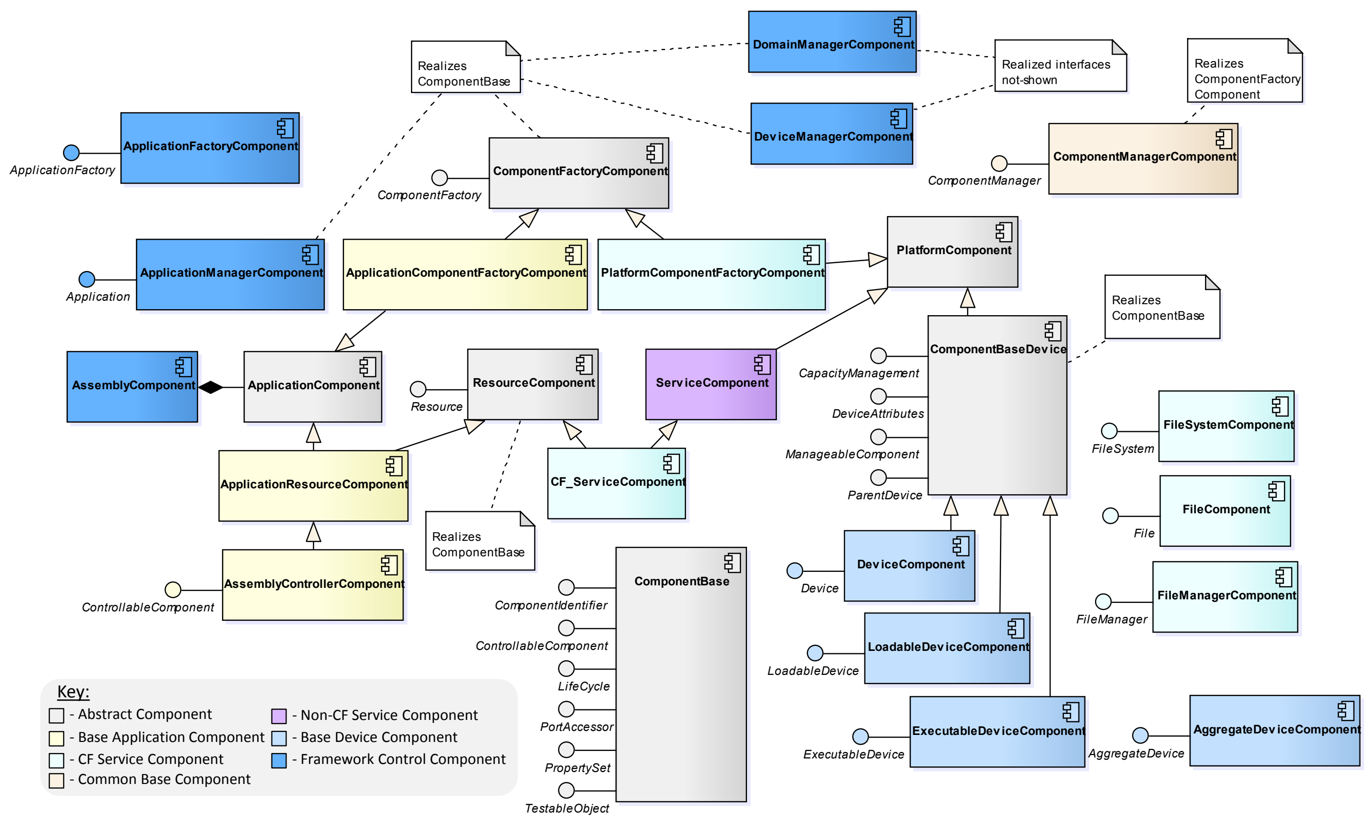


Figure 6 –SCA Component Hierarchy – Significant Components

VI. Expanded Features

SCA4 introduces an intra-application connection mechanism that allows the framework to connect multiple applications which may in turn share and exchange information. This capability permits SCA-based software defined radios and applications to support the deployment and interconnection of tactical mobile apps, such as those found in the U.S. Army's Marketplace, an Android-based app store. The SCA4's new connectivity options are ideal for handling communication to these external apps seamlessly via the Android presentation layer.

SCA4 also provides leniency on dictating the use of a specific middleware technology, namely the Common Object Request Broker Architecture (CORBA). The SCA no longer mandates the usage of CORBA as the sole middleware option. CORBA is still a viable alternative for SCA platforms and applications, but the SCA4 provides mechanisms to extend the specification with additional transfer mechanisms such as C++ RPC.

VII. Compliance Testing

The JTRS Test & Evaluation Laboratory (JTEL) is the test authority for Compliance Testing of the Software Communications Architecture (SCA). JTEL performs its compliance testing using a combination of manual processes and an automated tool, the JTRS Test Application (JTAP). An additional feature of the SCA4 specification development process was the steps that were taken to shorten the overall development life cycle by increasing the percentage of automated tests.

Automated testability was expanded in SCA4 through a combination of actions. The team performed a detailed assessment of the manually validated SCA 2.2.2 requirements. The assessment verified that the previous disposition, the requirement needed manual validation. Once the test method was confirmed the requirement was analyzed to ensure that it was a relevant and necessary at the SCA level. If the requirement was deemed non-essential then the text was either removed or it was refactored to provide development guidance. For instances where the first two tests passed, the requirement was evaluated to see if it could be reworded in a fashion that preserved the initial intent but made it testable. As existing requirements were modified or new requirements inserted within the specification, the team reviewed each change to insure that it could be tested.

VIII. Conclusion

The intent of SCA4 is to enhance the framework's ability to support program specific maintenance and development while mitigating impact to outstanding platforms. SCA4 takes the next step in streamlining the development and maintenance of software defined radios all while promoting flexibility and security as ingrained features. This newest standard, officially versioned as SCA 4.0, was approved by the JTRS Interface Control Working Group (ICWG) and the Wireless Innovation

Forum (WINNF) on February 28, 2012.

References

[1] *Software Communication Architecture*, JPEO JTRS, 4.0, 2012